

```

## We start with the requested time reporting points, in Times.
## Events are added with insertEvent(). insertEvent() first inserts
## the new event time(s) to the Eventtimes vector, then merges that/those
## time(s) into the Times vector. Finally, EventTimeindices, which
## is a pointer into the Times vector where Events must occur, is
## reconstructed. insertEvents() should only be used to schedule future
## events.
## 

mergeVector <- function(x,y) {
  sort(unique(c(x,y)))
}

ConstructEvents <-
function(times) {
  local({
    Times <- reqTimes <- sort(times)
    Eventtimes <- numeric(0)
    Events <- vector("list",0)
    Pulses <- new.env()
    Eventindx <- 1
    I <- 0
    Tint <- numeric(0)
    #EventTimeindices <- numeric(0)
    EventTimeindices <- c(1,length(Times))
    list(
      insertTime = function(when) {
        Times <<- sort(c(when, Times))
      },
      insertEvent = function(when, event, now){
        ## guard against setting events in the past, and duplicates in
        ## when (which would be an error)
        when <- unique(when[when > now])
        if (length(when) == 0) return()
        ## Insert 'whens' and events in Eventtimes and Events,
        ## respectively
        ##
        ## Only need to insert new 'whens' into Eventtimes
        dups <- when %in% Eventtimes
        if (any(!dups)) {
          Eventtimes <- c(Eventtimes,when[!dups])
          Events <- c(Events, rep(list(list(event)),sum(!dups)))
        }
        ## !!! The following looks wrong. 'dups' is like 'when',
        ## not like 'Events'
        if (any(dups)) {
          ETdups <- Eventtimes %in% when[dups]
          Events[ETdups] <-
            lapply(Events[ETdups],function(z) c(z,list(event)))
        }
        indx <- order(Eventtimes)
        Eventtimes <<- Eventtimes[indx]
        Events <<- Events[indx]
        ## rebuild Times to include Eventtimes
        ## mergeVector has a simple definition, but
        ## can perhaps be made more efficient with some work
      }
    )
  })
}

```

```

Times <- mergeVector(Times, when)
EventTimeindices <-
  if (length(Eventtimes) > 0) {
    findInterval(Eventtimes, Times)
  } else {
    c(1,length(Times))
  }
},
## CHANGED 2/11/09 JFW
doEvents = function(now,Parms,y,ydisc, ev)
{
  state <- list(y,ydisc)
  if (Eventindx <= length(Eventtimes))
  {
    if (now == Eventtimes[Eventindx])
    {
      events <- Events[[Eventindx]]
      for (i in 1:length(events)) {
        state <- events[[i]](now, Parm, state[[1]], state[[2]], ev )
      }
      Eventindx <- Eventindx + 1
    }
  }
  return(state)
},
getNextInterval = function(now) {
  if (l == 0) {
    ## First start
    l <- if (EventTimeindices[1] == 1) 2 else 1
    Tint <- Times[1:(EventTimeindices[l])]
  } else {
    ## There are two possibilities here:
    ## 1) everything went OK:
    if (now == Tint[length(Tint)]) {
      ## Then,
      if (l < length(EventTimeindices)) {
        Tint <- Times[EventTimeindices[l]:EventTimeindices[l+1]]
        l <- l + 1
      } else {
        Tint <- Times[EventTimeindices[l]:length(Times)]
      }
    } else { ## 2) Tint[1] < now < Tint[length(Tint)]
      Tint <- c(now, Tint[Tint > now])
    }
  }
  Tint
},
newPulse = function(pname) {
  assign(pname,1,envir=Pulses)
},
getthisPulse = function(pname) {
  j <- get(pname,envir=Pulses)
  assign(pname,j+1,envir=Pulses)
  j
},
quit = function(now) {

```

```
    now >= Times[length(Times)]  
  })  
}  
}
```